View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by International Institute for Applied Systems Analysis (IIASA)

International Institute for Applied Systems Analysis
IIASA www.iiasa.ac.at

A Standard Input Format for Computer Codes Which Solve Stochastic Programs with Recourse and a Library of Utilities to Simplify its Use

Edwards, J., Birge, J.R., King, A.J. and Nazareth, J.L.

IIASA Working Paper

WP-85-003

January 1985

Edwards, J., Birge, J.R., King, A.J. and Nazareth, J.L. (1985) A Standard Input Format for Computer Codes Which Solve Stochastic Programs with Recourse and a Library of Utilities to Simplify its Use. IIASA Working Paper. WP-85-003
Copyright © 1985 by the author(s). http://pure.iiasa.ac.at/2697/

Working Papers on work of the International Institute for Applied Systems Analysis receive only limited review. Views or opinions expressed herein do not necessarily represent those of the Institute, its National Member Organizations, or other organizations supporting the work. All rights reserved. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage. All copies must bear this notice and the full citation on the first page. For other purposes, to republish, to post on servers or to redistribute to lists, permission must be sought by contacting repository@iiasa.ac.at

NOT FOR QUOTATION
WITHOUT PERMISSION
OF THE AUTHOR

A STANDARD INPUT FORMAT FOR COMPUTER CODES WHICH SOLVE STOCHASTIC PROGRAMS WITH RECOURSE AND A LIBRARY OF UTILITIES TO SIMPLIFY ITS USE

Jonathan Edwards
John Birge
Alan King
Larry Nazareth

January 1985
WP-85-03

Working Papers are interim reports on work of the International Institute for Applied Systems Analysis and have received only limited review. Views or opinions expressed herein do not necessarily represent those of the Institute or of its National Member Organizations.

INTERNATIONAL INSTITUTE FOR APPLIED SYSTEMS ANALYSIS
2361 Laxenburg, Austria

ABSTRACT

We explain our suggestions for standardizing input formats for computer codes which solve stochastic programs with recourse. The main reason to set some conventions is to allow programs implementing different methods of solution to be used interchangeably. The general philosophy behind our design is a) to remain fairly faithful to the *de facto* standard for the statement of LP problems established by IBM for use with MPSX and subsequently adopted by the authors of MINOS, b) to provide sufficient flexibility so that a variety of problems may be expressed in the standard format, c) to allow problems originally formulated as deterministic LP to be converted to stochastic problems with a minimum of effort, d) to permit new options to be added as the need arises, and e) to provide some routines to facilitate the task of reading files specified in the standard format.

FOREWORD

Developing methodology and tools for optimal decision making under uncertainty was always a major part of research in System and Decision Sciences Area. For the last two years, the Adaptation and Optimization Project was involved in developing methods and computer implementations for one of the important parts of such methodology -- stochastic programming.

This paper is among those which describes one of the results of these efforts -- the collection of routines designed to solve stochastic programming problems. It contains a description of standardized input formats for some of the routines of this collection, and a library of utilities to simplify its use.

A.B. Kurzhanski
Chairman
System and Decision Sciences
Program

Introduction

Over the past six months, the Adaptive Optimization project of the Systems and Decision Sciences program at the International Institute for Applied Systems Analysis has collected a number of computer programs written to solve various problems in stochastic programming. Our goal has been to organize these codes so that they may be distributed on magnetic tape to researchers, who might benefit from having several algorithms with which to experiment. However, we have come to realize that the process of tinkering with the various methods will be greatly complicated because each program has its own format for input data. We have therefore developed a standard input format for stochastic programs with recourse. To encourage and simplify its use, we have based it on the input format developed by IBM for the extended Mathematical Programming Subsystem (MPSX) (IBM, 1972) and adopted by the authors of the Modular In-core Nonlinear Optimization System (MINOS) (Murtagh and Saunders, 1977) and we have written a number of low level subroutines to read files written in the standard format.

The Problem

The general form of the stochastic program with recourse is taken to be

$$\text{minimize } cx + Q(x) \tag{1}$$

subject to

$$Ax \left\{ \begin{matrix} \leq \\ = \\ \geq \end{matrix} \right\} b$$

$$l \leq x \leq u$$

where

$$Q(x) = \mathrm{E}_{\chi}\left\{ \min_{y \in Y} q(y,\chi) \mid \mathrm{T}(\chi)x + \mathrm{W}(\chi)y = p(\chi) \right\} .$$

x and y denote the decision and recourse variables, respectively, chi denotes an event, T(chi) and W(chi) denote the technology and recourse matrices, respectively, and $ roman E sub chi $ denotes expectation. In subsequent references to the technology matrix, the recourse matrix, the stochastic right hand side, $p(chi)$, and the penalty function, $q(y, chi)$, we omit the arguments y and chi.

Organization of the Data: Control, Core, and Stochastics Files

The data required by a program written to solve the stochastic program in (1) can be divided logically into three files: a control file, a "core" file, and a "stochastics" file. Roughly speaking, the control file contains any data particular to the program and the core and stochastics files contain the data that define the problem.

As its name implies, the control file contains any information that is used to guide the execution of the program. For example, the control file might include a limit on the number of steps permitted and a tolerance for convergence if the algorithm implemented in the program were iterative in nature, file name and unit number assignments if the program required several files, or upper limits on the amount of storage needed if the program allocated array space "dynamically." The control file also contains any information that must be read before the program profitably can read the contents of the matrices

and vectors that appear in the problem, e.g., the dimensions of those structures. Because the contents of the control file depend heavily on the algorithm employed and the manner in which it is implemented, we have not included a standard format for control files. Indeed, the rigid structure of the format we propose (particularly its strict use of specific columns as field delimiters) makes it unsuitable for application to files whose contents are liable to change frequently.

The core file contains the bounds on the decision vector, x, the contents of the matrices by which it is multiplied, and the contents and ranges of the rows of the deterministic right hand side vector, b. The core file for a stochastic LP thus corresponds in large measure to the data file that MPSX or MINOS would require to solve the equivalent nonstochastic LP (i.e., the same problem with $Q(x)$ removed).

The stochastics file defines the technology matrix, T, the distribution of the rows of the stochastic right hand side vector, p, the contents of the recourse matrix, W, and the function q. We have chosen to partition the input in this fashion so that a problem originally formulated as a linear program and expressed in standard MPSX format may be augmented later by a stochastics file, thereby permitting certain elements (e.g., the right hand side) to be stochastic.

Overview of the Standard Input Format

The proposed format is quite similar to the MPSX format, which is described on pages 199 through 209 of (IBM, 1972), although there are some differences. As in the MPSX format, each data file contains a number of sections, some of which are optional. A "header line" (or "header") marks the beginning of each section†. Most sections contain data lines. A data line is divided into six fields, some of which may be empty. Specific columns delineate field boundaries. There are three name fields, two numeric fields, and a code field. The columns that constitute these fields are

- columns 2 and 3: code field
- columns 5 through 12: first name field
- columns 15 through 22: second name field
- columns 25 through 36: first numeric field
- columns 40 through 47: third name field
- columns 50 through 61: second numeric field

(all column ranges are inclusive). Comment lines contain an asterisk (*) in the first column and may appear anywhere.

Unlike the MPSX format, names may contain imbedded blanks or leading blanks (although this last is not recommended). The contents of the name fields are interpreted as character strings, so names may begin with a digit. All lower case letters in the code and name fields are translated to their upper case equivalents. Values in the numeric fields must contain a decimal point. The MPSX convention concerning comments following a dollar sign ($) in the first column of the second or third name fields has not been adopted as part of the standard format.

Following are descriptions of each of the data files. Each description contains a list of the sections that constitute the corresponding data file. These

† (IBM, 1972, p. 199) uses the term "indicator card" rather than header.

sections must appear in the data file in the same order as they appear in the list, although sections marked "optional" need not appear at all.

The Core File

The core file specifies

- the linear portion of the objective, c,
- the contents of the constraint matrix, A, and possibly the contents of the technology matrix, T, and of the recourse matrix, W,
- the deterministic right hand side, b,
- the bounds on the decision vector, x, and
- the ranges on the right hand side.

The core file contains the following sections: "NAME," "ROWS," "COLUMNS," "RHS," "RANGES," "BOUNDS," and "ENDATA." These sections assume more or less the same role in the standard format as they do in the MPSX format. Therefore, we give only an abbreviated description of these sections and note differences between the standard format and the MPSX format.

1. **NAME** - This is an informative header line (the section contains no data lines). The user may enter any characters desired in columns 15 through 72 (the MPSX format restricts names to eight alphanumeric characters).
2. **ROWS** - As in the MPSX format, this section specifies the names of the rows of A, the name of the row in the COLUMNS section that contains the elements of c, and the type of constraint (equality or inequality) represented by each row. In some cases, this section also specifies the names of the rows of T. Rows formed by a linear combination of two other rows (type "D" rows) and scaling of rows (use of the "'SCALE'" keyword) are supported in the MPSX format but are not permitted in the standard format.
3. **COLUMNS** - As in the MPSX format, this section specifies the names of the columns of A and of c and contains the values of the nonzero elements of A and of c. In some cases, this section also specifies the names of the columns of W, contains the nonzero elements of W, and/or contains the nonzero elements of T. Scaling of columns (use of the "'SCALE'" keyword) is supported in the MPSX format but is not permitted in the standard format.
4. **RHS** - This section specifies the names of the rows of b and contains the values of the nonzero elements of b. This section is identical to its counterpart in the MPSX format.
5. **RANGES** (optional) - This section specifies the ranges on the rows of b. This section is identical to its counterpart in the MPSX format.
6. **BOUNDS** (optional) - This section specifies the bounds on the rows of the decision vector, x. This section is identical to its counterpart in the MPSX format.
7. **ENDATA** - This line marks the end of the core file (the section contains no data lines) and is identical to its counterpart in the MPSX format.

The Stochastics File

The stochastics file specifies

- the contents of the technology matrix, T,

- the distribution of the stochastic right hand side, p,
- the contents of the recourse matrix, W, and
- the form of the penalty function, q.

The stochastics file contains the following sections: "NAME," "TECHNOLOGY," "DISTRIBUTIONS," "RECOURSE," "OBJECTIVES," and "ENDATA." After the "OBJECTIVES" section additional sections may appear containing data particular to a given algorithm. A program should read only those sections it needs from the file and should ignore the rest.

Most sections may take one of several forms, and the user must enter the name of one of them beginning in column 15 of the header line. A description of each of the sections, the forms they may assume, and their contents follows.

1. **NAME** - This is an informative header line (the section contains no data lines). The user may enter any characters desired in columns 15 through 72.
2. **TECHNOLOGY** - This section specifies the contents of T. The section may take one of the forms whose names follow:

   DETERMINISTIC (the elements of T follow) - The technology matrix is given by the data following the section header. The format of the data is identical to that of the COLUMNS section of the core file, i.e., the contents of the matrix are specified in column order. The first name field on a line (columns 5 through 12) contains the name of the column. The remaining name/numeric field pairs (columns 15 through 22/25 through 36 and 40 through 47/50 through 61) specify a row name and the contents of the matrix at the position given by the row and column names. The row names form a subset of the row names in the ROWS section of the core file.

   CORE (the elements of T appear in the core file) - The data consists of a list of names which form a subset of the names specified in the ROWS section of the core file. The contents of these rows (as specified in the COLUMNS section of the core file) constitute the technology matrix. One name appears per line, in the first name field (columns 5 through 12).

   STOCHASTIC (the elements of T are supplied by a subroutine) - The data consists of a list of the names of the rows of the technology matrix. Each row name has associated with it one or more column names. The column names specify the active columns within the given row and form a subset of the column names specified in the COLUMNS section of the core file. The values for the technology matrix do not appear in either data file but are supplied by a subroutine written by the user. The row names appear in the first name field of a line (columns 5 through 12) and the other two name fields (columns 15 through 22 and 40 through 47) are available for the column names.

   NONE (no data) - There is no data. The user must decide where and how to obtain the necessary values.
3. **DISTRIBUTIONS** - This section specifies the distribution of the rows of p. The section may take one of the forms whose names follow:

   DISCRETE (each row is independently distributed) - Each row of p may take one of a fixed number of values. The data for this form consists of a number of "definitions," which are analogous to the "vectors" in the

RANGES and BOUNDS sections of the core file (see IBM, 1972). Each definition specifies the distribution of every row of p and consists of a number of sets of entries of the form "defname rowname value probability." Within a given definition, there is one such set for each of the rows named in the TECHNOLOGY section. "defname" is the name of the definition to which the entry belongs; it occupies the first name field on a line (columns 5 through 12). "rowname" is the name of the row associated with the entry; it occupies the second name field on a line (columns 15 through 22). "value" and "probability" are a value for the row and its likelihood, respectively. They occupy the first and second numeric fields (columns 25 through 36 and 50 through 61), respectively.

The sum of the probabilities for a given row must be unity. The values specified for a given row must be distinct. Entries for different rows or different definitions must not be mixed together in the input file.

As an example, let the T matrix have two rows, TROW1 and TROW2, and define two distributions for the rows of p as follows:

$$\text{Row 1} = \begin{Bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{Bmatrix} \text{ with probability } \begin{Bmatrix} 0.4 \\ 0.2 \\ 0.2 \\ 0.2 \end{Bmatrix} \qquad (1)$$

$$\text{Row 2} = \begin{Bmatrix} 8 \\ 9 \\ 0 \end{Bmatrix} \text{ with probability } \begin{Bmatrix} 0.6 \\ 0.3 \\ 0.1 \end{Bmatrix}$$

and

$$\text{Row 1} = \begin{Bmatrix} 2 \\ 4 \end{Bmatrix} \text{ with probability } \begin{Bmatrix} 0.5 \\ 0.5 \end{Bmatrix} \qquad (2)$$

$$\text{Row 2} = 2 \text{ with probability } 1.0 \ .$$

The contents of the name and numeric fields for these distributions are shown in table 1. The user specifies which is the desired definition (our definition names "DIST1" and "DIST2" were chosen arbitrarily) when the appropriate input utility is called. Note that every value contains a decimal point.

First Name Field	Second Name Field	First Numeric Field	Second Numeric Field
DIST1	TROW1	1.0	0.4
DIST1	TROW1	2.0	0.2
DIST1	TROW1	3.0	0.2
DIST1	TROW1	4.0	0.2
DIST1	TROW2	8.0	0.6
DIST1	TROW2	9.0	0.3
DIST1	TROW2	0.0	0.1
DIST2	TROW1	2.0	0.5
DIST2	TROW1	4.0	0.5
DIST2	TROW2	2.0	1.0

Table 1. Contents of a sample DISCRETE DISTRIBUTIONS section.

SIMULATION (the rows are supplied by a subroutine) - There are no data lines in this case. The program obtains its values from a subroutine written by the user.

PIECEWISE (piecewise constant pdf) - Each row of p takes a value within one of a finite number of ranges. Within a range, all values are equally likely. However, within a set of ranges, all ranges are *not* equally likely. The data for this form consists of a number of "definitions," which are analogous to the "vectors" in the RANGES and BOUNDS sections of the core file (see IBM, 1972). Each definition specifies the distribution of every row of p and consists of a number of sets of entries of three lines each. Within a given definition, there is one such set for each of the rows named in the TECHNOLOGY section. Each three line entry within a set describes a range for the row associated with the set. The first line in an entry contains the letters "PC" in the code field (columns 2 and 3), the name of the definition to which the entry belongs in the first name field (columns 5 through 12), the name of the row with which this range is associated in the second name field (columns 15 through 22), and the probability that the row takes a value within the range in the first numeric field (columns 25 through 36). The second and third lines in an entry specify the upper and lower bounds of the range. For both bounds, the code field contains the letters "BD", the first name field contains the name of the definition to which the entry belongs, the second name field contains the name of the row with which the range is associated, and the first numeric field contains the bound value.

The sum of the probabilities for the ranges for a given row must be unity. Entries for different rows, different ranges, or different definitions must not be mixed together in the input file.

As an example, let the T matrix have two rows, TROW1 and TROW2, and define two distributions for the rows of p as follows:

$$\text{Row 1 in } \begin{cases}[1,2] \\ [3,4]\end{cases} \text{ with probability } \begin{cases}0.6 \\ 0.4\end{cases} \qquad (1)$$

$$\text{Row 2 in } \begin{cases}[5,7] \\ [1,3] \\ [0,1]\end{cases} \text{ with probability } \begin{cases}0.7 \\ 0.1 \\ 0.2\end{cases}$$

and

$$\text{Row 1 in } \begin{cases}[2,4] \\ [5,9]\end{cases} \text{ with probability } \begin{cases}0.5 \\ 0.5\end{cases} \qquad (2)$$

Row 2 in [2,3] with probability 1.0 .

The contents of the code, name and numeric fields for these distributions are shown in table 2. The user specifies which is the desired definition (our definition names "DIST1" and "DIST2" were chosen arbitrarily) when the appropriate input utility is called. Note that every value contains a decimal point.

Code Field	First Name Field	Second Name Field	First Numeric Field
PC	DIST1	TROW1	0.6
BD	DIST1	TROW1	1.0
BD	DIST1	TROW1	2.0
PC	DIST1	TROW1	0.4
BD	DIST1	TROW1	3.0
BD	DIST1	TROW1	4.0
PC	DIST1	TROW2	0.7
BD	DIST1	TROW2	5.0
BD	DIST1	TROW2	7.0
PC	DIST1	TROW2	0.1
BD	DIST1	TROW2	1.0
BD	DIST1	TROW2	3.0
PC	DIST1	TROW2	0.2
BD	DIST1	TROW2	0.0
BD	DIST1	TROW2	1.0
PC	DIST2	TROW1	0.5
BD	DIST2	TROW1	2.0
BD	DIST2	TROW1	4.0
PC	DIST2	TROW1	0.5
BD	DIST2	TROW1	5.0
BD	DIST2	TROW1	9.0
PC	DIST2	TROW2	1.0
BD	DIST2	TROW2	2.0
BD	DIST2	TROW2	3.0

Table 2. Contents of a sample PIECEWISE DISTRIBUTIONS section.

SCENARIOS (the value of p is defined by a sample of vectors) - The p vector may take one of a finite number of values. The data for this form consists of a number of "definitions," which are analogous to the "vectors" in the RANGES and BOUNDS sections of the core file (see IBM, 1972). Each definition provides a sample of vectors and consists of sets of entries giving a value for p and the probability that p takes that value. The first line in each entry contains the letters "SC" in the code field (columns 2 and 3), the name of the definition to which the entry belongs in the first name field (columns 5 through 12), a name identifying the scenario in the second name field (columns 15 through 22), and the probability that p takes the value associated with this scenario in the first numeric field (columns 25 through 36). Subsequent lines specify the values that the rows of p assume under the scenario. There must be one of these lines for each row

named in the TECHNOLOGY section. The code field of these lines contains the letters "RV", the first name field contains the name of the defninition to which the entry belongs, the second name field contains the name of the row whose value the line specifies, and the first numeric field contains the value.

The sum of the probabilities for the scenarios in a given definition must be unity. Entries for different scenarios or different definitions must not be mixed together in the input file.

As an example, let the T matrix have two rows, TROW1 and TROW2, and define two distributions of the vector p as follows:

$$\text{Vector} = \begin{cases} [1\ 2] \\ [3\ 4] \\ [5\ 6] \end{cases} \text{ with probability } \begin{cases} 0.5 \\ 0.3 \\ 0.2 \end{cases} \tag{1}$$

and

$$\text{Vector} = \begin{cases} [1\ 2] \\ [1\ 5] \end{cases} \text{ with probability } \begin{cases} 0.5 \\ 0.5 \end{cases} \tag{2}$$

The contents of the code, name and numeric fields for these distributions are shown in table 3. The user specifies which is the desired definition (our definition names "SAMP1" and "SAMP2" were chosen arbitrarily) when the appropriate input utility is called. The scenario names "SCEN1," "SCEN2," and "SCEN3" where chosen arbitrarily. Note that every value contains a decimal point.

NONE (no data) - There is no data. The user must decide where and how to obtain the necessary values.

4. **RECOURSE** - This section specifies the contents of W. The section may take one of the forms whose names follow:

SIMPLE (simple recourse) - There are no data lines in this case. The recourse matrix is assumed to be [I,-I], where I has rank equal to the number of rows in the technology matrix.

DETERMINISTIC (the elements of W follow) - The recourse matrix is given by the data following the section header. The format of the data is identical to that of the COLUMNS section of the core file, i.e., the contents of the matrix are specified in column order. The first name field on a line (columns 5 through 12) contains the name of the column. The remaining name/numeric field pairs (columns 15 through 22/25 through 36 and 40 through 47/50 through 61) specify a row name and the contents of the matrix at the position given by the row and column names. The row names form a subset of the row names in the TECHNOLOGY section.

CORE (the elements of W appear in the core file) - The data consists of a list of names which form a subset of the column names specified in the COLUMNS section of the core file. The contents of those columns (as specified in the COLUMNS section of the core file) constitute the recourse matrix. One name appears per line, in the first name field (columns 5

Code Field	First Name Field	Second Name Field	First Numeric Field
SC	SAMP1	SCEN1	0.5
RV	SAMP1	TROW1	1.0
RV	SAMP1	TROW2	2.0
SC	SAMP1	SCEN2	0.3
RV	SAMP1	TROW1	3.0
RV	SAMP1	TROW2	4.0
SC	SAMP1	SCEN3	0.2
RV	SAMP1	TROW1	5.0
RV	SAMP1	TROW2	5.0
SC	SAMP2	SCEN1	0.5
RV	SAMP2	TROW1	1.0
RV	SAMP2	TROW2	2.0
SC	SAMP2	SCEN2	0.5
RV	SAMP2	TROW1	1.0
RV	SAMP2	TROW2	5.0

Table 3. Contents of a sample SCENARIOS DISTRIBUTIONS section.

through 12).

STOCHASTIC (the elements of W are supplied by a subroutine) - The data consists of a list of the names of the rows of the recourse matrix. Associated with each name is one or more column names. These column names specify the active columns within the given row and form a subset of the column names specified in the COLUMNS section of the core file. The values for the recourse matrix do not appear in either data file but are supplied by a subroutine written by the user. The row names appear in the first name field of a line (columns 5 through 12) and the other two name fields (columns 15 through 22 and 40 through 47) are available for the column names.

NONE (no data) - There is no data. The user must decide where and how to obtain the necessary values.

5. **OBJECTIVES** - This section specifies the form of q. The section may take one of the forms whose names follow:

LINEAR (q is a linear function) - The recourse objective is given by $q(y) = qy$, where q is given by the data following the section header. The data for this form consists of a number of "definitions," which are analogous to the "vectors" in the RANGES and BOUNDS sections of the core file (see IBM, 1972). Each definition specifies the elements of q and consists of entries of the form "defname name value," where "defname" is the name of the definition to which the entry belongs, "name" is the name of a column of W (or of a row of T; see below) and "value" is the value for the corresponding row of q. "defname" occupies the first name field on a line

(columns 5 through 12), "name" occupies the second name field (columns 15 through 22) and "value" occupies the first numeric field (columns 25 through 36).

Entries for different definitions must not be mixed together in the input file.

As an example, let the W matrix have two columns, WCOL1 and WCOL2, and define two vectors q as follows:

$$q = [7\ 9] \tag{1}$$

and

$$q = [3\ 3] \tag{2}$$

The contents of the name and numeric fields for these vectors are shown in table 4. The user specifies which is the desired definition (our definition names "VEC1" and "VEC2" were chosen arbitrarily) when the appropriate input utility is called. Note that every value contains a decimal point.

First Name Field	Second Name Field	First Numeric Field
VEC1	WCOL1	7.0
VEC1	WCOL2	9.0
VEC2	WCOL1	3.0
VEC2	WCOL2	3.0

Table 4. Contents of a sample LINEAR OBJECTIVES section.

PIECEWISE (q is two-piece linear) - The recourse objective is assumed to be two-piece continuous about zero, i.e.

$$q(y) = \sum_i q_i(y_i) \text{ with } q_i(y_i) = \begin{cases} -q_i^+ y_i & , \ y_i \le 0 \\ q_i^- y_i & , \ y_i \ge 0 \end{cases}$$

The data for this form consists of a number of "definitions," which are analogous to the "vectors" in the RANGES and BOUNDS sections of the core file (see IBM, 1972). Each definition specifies the values of q_i^+ and q_i^- for all i and consists of entries of the form "defname name value value", where "defname" is the name of the definition to which the entry belongs, "name" is the name of a column of W (or of a row of T; see below), the first value gives the corresponding value of q^+, and the second value gives the corresponding value of q^-. The names occupy the first and second name fields on a line (columns 5 through 12 and 15 through 22) and the values occupy the first and second numeric fields (columns 25 through 36 and 50 through 61).

Entries for different definitions must not be mixed together in the input file.

As an example, let the W matrix have two columns, WCOL1 and WCOL2, and define two vectors q as follows:

$$q = [\begin{cases} -2 & , \ y_1 \leq 0 \\ 5 & , \ y_1 \geq 0 \end{cases} \begin{cases} -3 & , \ y_2 \leq 0 \\ 7 & , \ y_2 \geq 0 \end{cases}] \qquad (1)$$

and

$$q = [\begin{cases} -5 & , \ y_1 \leq 0 \\ 3 & , \ y_1 \geq 0 \end{cases} \begin{cases} -9 & , \ y_2 \leq 0 \\ 2 & , \ y_2 \geq 0 \end{cases}] \qquad (2)$$

The contents of the name and numeric fields for these vectors are shown in table 5. The user specifies which is the desired definition (our definition names "VEC1" and "VEC2" were chosen arbitrarily) when the appropriate input utility is called. Note that every value contains a decimal point and that the values of q_i^+ are positive.

First Name Field	Second Name Field	First Numeric Field	Second Numeric Field
VEC1	WCOL1	2.0	5.0
VEC1	WCOL2	3.0	7.0
VEC2	WCOL1	5.0	3.0
VEC2	WCOL2	9.0	2.0

Table 5. Contents of a sample OBJECTIVES (PIECEWISE) section.

NONE (no data) - There is no data. The user must decide where and how to obtain the necessary values.

Note - if the recourse matrix is simple (i.e., if there are no column names for W), row names of T are substituted for column names of W in the OBJECTIVES section.

6. **ENDATA** - This line marks the end of the stochastics file (the section contains no data lines).

It is clear that we have covered only a few of the possibilities for most of the above sections. However, the format is such that new forms can be added as the need arises.

Standard Data Structures

One of the benefits of a standard format is that a library of routines to facilitate the task of reading files specified in the format may be written. We now describe some special data structures used by our utilities.

Packed Vectors and Matrices

The matrices and vectors that appear in a stochastic program with recourse are often sparse. Our utilities therefore attempt to conserve memory by saving only the nonzero elements of a matrix or vector. We call this a "packed" representation and refer to matrices or vectors stored in this fashion as "packed matrices" and "packed vectors," respectively.

A packed vector consists of two arrays†. One contains the nonzero vector elements. The other holds the row or column indices of the nonzero vector elements. We call these arrays the "values" array and the "indices" array, respectively. If the nonzero elements of a vector are $[v_{i_1}, \ldots, v_{i_n}]$ and are ordered so that $i_j < i_{j+1}$, then the jth element of the values array contains the value of v_{i_j} and the jth element of the indices array contains the value of i_j. The values and indices arrays for the vector [1,0,4,0,0,5] are shown in figure 1.

values	indices
1	1
4	3
5	6

Figure 1. Packed representation of the vector [1,0,4,0,0,5]

Just as an ordinary matrix may be viewed as a collection of vectors, a packed matrix may be thought of as a collection of packed vectors. A packed matrix consists of three arrays. One contains the the nonzero matrix elements. Another holds the row indices of the nonzero matrix elements. These two arrays are analogous to the values and indices arrays mentioned in the preceding paragraph and so are called the values array and the indices array, respectively. The third array organizes the matrix by column: the ith element of this array is the location in the values and indices arrays where the entries for the ith column of the matrix begin. Because the elements of the third array "point" to the matrix columns, we call this array the "pointers" array. If a matrix has n columns, the (n+1)th element of the pointers array contains the index of the element in the values array immediately following the last nonzero matrix element (i.e., the (n+1)th element points to the space immediately following the matrix entries). The values, indices, and pointers arrays for the 6×5 matrix

$$\begin{bmatrix} 0 & 6 & 1 & 4 & 2 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 4 & 0 & 0 & 0 & 3 \\ 3 & 1 & 0 & 3 & 4 \\ 0 & 0 & 5 & 0 & 5 \end{bmatrix}$$

† Throughout this paper, we use the term "array" to refer to "a nonempty sequence of data" (ANSI, 1978, p. 5-1) within the computer's memory. An array may have several dimensions

are shown in figure 2. The double lines in that figure mark the beginning of the entries for each column. Note that the third column vector of the matrix is identical to the vector whose packed representation is shown in figure 1 and as a result that the portions of the values and indices arrays that contain the entries for that vector in figure 2 are identical to the values and indices arrays in figure 1. Note also that the value of the first element of the pointers array is always 1.

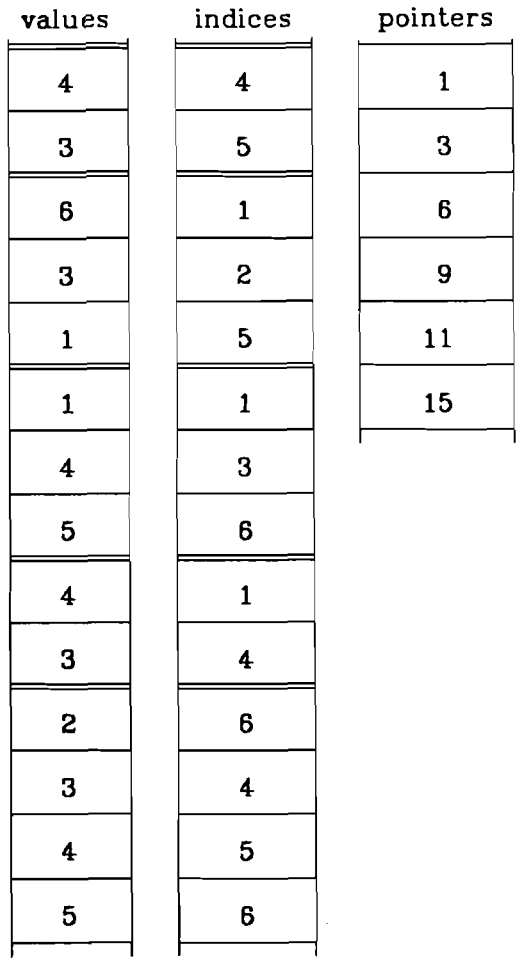

Figure 2. Packed representation of a 6×5 matrix

It is difficult to obtain a single row vector from the data structure shown in figure 2. To permit rapid access to row vectors, we add three arrays to a packed matrix and produce an "augmented packed matrix." The three arrays are called the "augmented pointers" array, the "augmented indices" array, and the "map" array. The augmented pointers array and the augmented indices array are analogous to the indices and pointers arrays of the packed matrix. The augmented indices array contains the column indices of the nonzero matrix elements and the augmented pointers array holds pointers to the beginning of the entries for each row. However, there is a small complication. The contents of the augmented pointers array cannot be used to access the values array directly since the entries in the values array are organized by column. The

or only one dimension. We reserve the word "matrix" to refer to the matrices in the problem, i.e., A, T, and W.

function of the map array is to map the augmented pointer array and the augmented indices array onto the values array (hence its name). Each element in the map array is the location in the values array of the matrix element whose row and column indices are defined by the appropriate entries in the augmented pointers array and the augmented indices array. Figure 3 shows the augmented pointers, augmented indices, map, and values arrays for the matrix used to provide the example in figure 2. The double lines in figure 3 mark the beginning of the entries for each row. Like the pointers array, the augmented pointers array always has the value 1 in its first element.

augmented pointers	augmented indices	map	values
1	2	3	4
5	3	6	3
6	4	9	6
7	5	11	3
9	2	4	1
13	3	7	1
15	1	1	4
	5	12	5
	1	2	4
	2	5	3
	4	10	2
	5	13	3
	3	8	4
	5	14	5

Figure 3.
Arrays for rapid access of row vectors
in the packed representation of a 6×5 matrix

The reader may be forgiven for wondering how the use of six arrays to represent a single matrix can possibly save space. Indeed, it might appear that the density of the matrix would have to be less than 0.16 for such an approach to conserve memory. The true picture is brighter because only the values array need be capable of expressing real numbers. The contents of the auxiliary arrays (indices, pointers, etc.) are integers. In the utilities we have written, the values array is double precision and the auxiliary arrays are half integer (integer*2). Table 6 shows the densities at which it becomes uneconomical to use the various packed representations.

Representation	Unpacked Dimensions	Savings realized if density < Exact	Approx. %
packed vector	n	$\frac{8}{10}$	80
packed matrix	m×n	$\frac{8}{10} - \frac{2}{10m} - \frac{2}{10mn}$	73
augmented matrix	m×n	$\frac{8}{14} - \frac{2(n+m)}{14mn} - \frac{4}{14mn}$	46

Table 6. Critical densities for packed data structures.

Since many of our input utilities use packed representations exclusively, we have provided routines to convert packed structures to their unpacked equivalents and vice versa. It is possible to write subroutines that use these utilities and the standard input utilities to read dense matrices without permanently consuming the additional memory that the packed representation requires. For example, to read the contents of an m×n matrix, M, whose density is greater than 0.73, into an m×n array, A, the user writes a subroutine and passes it the address of A (i.e., "call subroutine(A,...)"). The subroutine

- declares the pointers, indices, and values arrays for a packed matrix,
- calls the appropriate input utility to read the contents of M into the packed matrix, and
- calls the "expand matrix" utility, directing it to transfer the contents of the packed matrix into A.

The effect of these operations is to place the elements of M into A. The arrays used to represent the packed matrix disappear when the subroutine executes its "return" statement.

Stochastic Vectors

The stochastic right hand side vector, p, cannot easily be represented as a single packed vector. The DISTRIBUTIONS section (which defines p) may appear in several forms, each of which provides different information. Accordingly, there are three separate (but similar) data structures to describe the distribution of the stochastic right hand side.

The three data structures share a common organization: each has a "pointers" array, a "probabilities" array, and at least one "values" array. The contents of these arrays are interpreted differently for each data structure.

For the DISCRETE form of the DISTRIBUTIONS section, the values array contains the values that each row may assume and each entry in the probabilities array contains the likelihood that the row assumes the corresponding value. In this case, the pointers array plays a role similar to the augmented pointers array in an augmented packed matrix: it organizes the values and probabilities arrays by row. The ith entry in the pointers array is the location in the values and probabilities arrays where the entries for the ith row begin. Like the last element of the pointers array in a packed matrix, the last element of the pointers array in a stochastic vector points to the first "empty" entry in the values and probabilities arrays. The pointers, values, and probabilities arrays for the vector whose rows have the distribution

$$\text{Row } 1 = \begin{cases} 1 \\ 2 \\ 3 \\ 4 \end{cases} \text{ with probability } \begin{cases} 0.4 \\ 0.2 \\ 0.2 \\ 0.2 \end{cases}$$

$$\text{Row 2} = \begin{Bmatrix}5\\6\\7\end{Bmatrix} \text{ with probability } \begin{Bmatrix}0.1\\0.7\\0.2\end{Bmatrix}$$

$$\text{Row 3} = \begin{Bmatrix}2\\4\end{Bmatrix} \text{ with probability } \begin{Bmatrix}0.5\\0.5\end{Bmatrix}$$

$$\text{Row 4} = 3 \text{ with probability } 1.0$$

are shown in figure 4. The double lines in that figure mark the beginning of the entries for each row.

pointers	values	proba-bilities
1	1	0.4
5	2	0.2
8	3	0.2
10	4	0.2
11	5	0.1
	6	0.7
	7	0.2
	2	0.5
	4	0.5
	3	1.0

Figure 4.
Representation of a stochastic vector
(DISCRETE DISTRIBUTIONS).

The situation is quite similar for the PIECEWISE form of the DISTRIBUTIONS section, except that two values arrays are needed. One (the "high values" array) contains the upper endpoint for each interval. The other (the "low values" array) contains the lower endpoint for each interval. Each entry in the probabilities array contains the likelihood that the appropriate row assumes a value in the corresponding interval. The pointers array again organizes the other arrays by row. The pointers, high and low values, and probabilities arrays for the vector whose rows have the distribution

$$\text{Row 1 in } \begin{Bmatrix}[1,1.5]\\[2,2.5]\\[3,3.5]\\[4,4.5]\end{Bmatrix} \text{ with probability } \begin{Bmatrix}0.4\\0.2\\0.2\\0.2\end{Bmatrix}$$

$$\text{Row 2 in } \begin{Bmatrix}[5,5.3]\\[6,6.3]\\[7,7.3]\end{Bmatrix} \text{ with probability } \begin{Bmatrix}0.1\\0.7\\0.2\end{Bmatrix}$$

$$\text{Row 3 in } \begin{cases}[2,2.1] \\ [4,4.1]\end{cases} \text{ with probability } \begin{cases}0.5 \\ 0.5\end{cases}$$

$$\text{Row 4 in } [3,3.2] \text{ with probability } 1.0$$

are shown in figure 5. The double lines in that figure mark the beginning of the entries for each row. Note that the ranges were obtained by taking the values from the example shown in figure 4 as the lower bounds for the intervals and thus that the pointers and probabilities arrays are the same as in figure 4 and that the low values array is the same as the values array in figure 4.

pointers	low values	high values	proba-bilities
1	1	1.5	0.4
5	2	2.5	0.2
8	3	3.5	0.2
10	4	4.5	0.2
11	5	5.3	0.1
	6	6.3	0.7
	7	7.3	0.2
	2	2.1	0.5
	4	4.1	0.5
	3	3.2	1.0

Figure 5.
Representation of a stochastic vector
(PIECEWISE DISTRIBUTIONS).

The data structure for the SCENARIOS form of the DISTRIBUTIONS section is quite different from that for the DISCRETE and PIECEWISE forms. The values array, as always, contains the values that each row may assume. However, the values are grouped by scenario: the ith element of the pointers array is the location in the values array where the entries for the ith observation of the vector begin and the ith element of the probabilities array contains the likelihood that the vector assumes the corresponding value. If there are n observations, the (n+1)th entry in the pointers array is the index of the first unused element in the values array. The pointers, values, and probabilities arrays for the vector whose distribution is

$$\text{Vector} = \begin{cases}[1\ 2\ 3\ 4] \\ [3\ 5\ 2\ 1] \\ [0\ 2\ 5\ 0] \\ [9\ 4\ 3\ 6]\end{cases} \text{ with probability } \begin{cases}0.3 \\ 0.3 \\ 0.2 \\ 0.2\end{cases}$$

are shown in figure 6. The double lines in that figure mark the beginning of the

entries for each scenario.

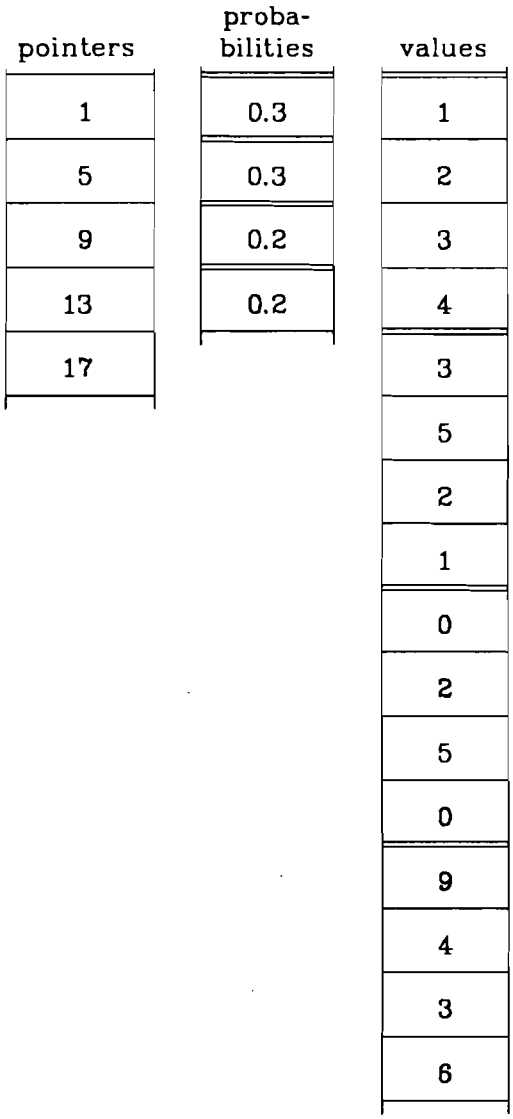

Figure 6.
Representation of a stochastic vector
(SCENARIOS DISTRIBUTION).

Since each observation has the same number of elements (i.e., the number of rows in p), the pointers array is not strictly necessary. We have included it to provide some uniformity in the components of the data structures for each form of the stochastic right hand side.

Simple Lists and Hash Tables

Lists play an important role in the standard input format. Some sections (e.g., the ROWS section and the CORE form of the TECHNOLOGY section) are little more than a list of names. Other sections (e.g., the COLUMNS section and the OBJECTIVES section) require that a list of names be searched. Our input utilities are able to represent these lists in either of two data structures.

A list of names may be kept in a "simple list" or in a "hash table" (Knuth, 1973). A simple list is essentially a one-dimensional array of character strings.

New entries are placed in the first available element and the list is searched sequentially. The hashing process uses a function to assign an index to each name; the name is then placed in the corresponding element of the hash table. Searches are therefore faster when a hash table is used, but a hash table requires more space than a simple list. The simple list and a hash table generated when the names "row00001," "row00002," ..., "row00006" are entered in order are shown in figure 7. Also shown is the location where the name "row00007" would be entered. The hash function used to generate the indices for the hash table appears in our "pchash" utility.

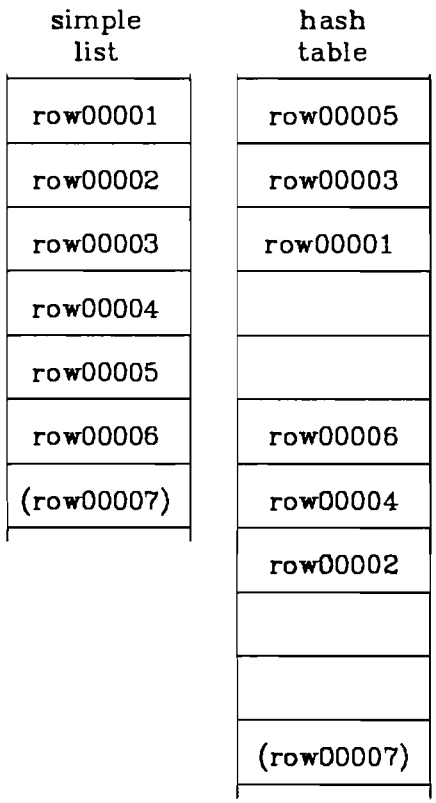

Figure 7. Simple list and hash table

Packed Matrices and User-Written Routines

To use the STOCHASTIC form of the TECHNOLOGY and the RECOURSE sections or the SIMULATIONS form of the DISTRIBUTIONS section of the stochastics file, the user must write a routine to supply the values of the active elements of the corresponding matrix or vector. By convention, the first parameter passed to this routine is the values array for the appropriate structure. All the user's routine need do is enter its data in column order into this array. It is the calling routine's responsibility to establish the contents of any auxiliary arrays.

Overview of the Utilities

Nazareth (1982) discusses the development of subroutines which may be used "as the primitives or basic operators of a language for implementing LP algorithms." His thesis is that a mathematician should develop algorithms to solve problems, not grind out FORTRAN routines to read input files. The purpose of our utilities is to free the program author from the tyranny of minutiae and to allow the researcher to concentrate on methods rather than on means.

The ancestors of our routines are the modules in the LPKIT collection (Nazareth, 1982). Accordingly, we have adopted LPKIT's naming conventions. "Problem oriented modules" have names which begin with the letter "p." Our problem oriented modules read the sections defined in the standard input format and perform various other chores. "Algorithm oriented modules" have names which begin with the letter "a." The algorithm oriented modules in our collection operate on packed data structures; these subroutines therefore have names which begin with the letters "ad." Tables 7 through 10 contain a summary of our utilities.

Subroutine	Function
adcmat	Produce a packed matrix from a matrix of full rank (i.e., remove zeros)
adcvec	Produce a packed vector from a vector of full rank (i.e., remove zeros)
adxmat	Produce a matrix of full rank from a packed matrix (i.e., restore zeros)
adxvec	Produce a vector of full rank from a packed vector (i.e., restore zeros)
admkax	Produce an augmented packed matrix from a packed matrix (i.e., generate auxiliary arrays)

Table 7. Routines that manipulate packed data structures

Subroutine	Function
prcrow	Read the ROWS section of a core file
prcbou	Read the BOUNDS section of a core file
prsdis	Read the DISCRETE form of the DISTRIBUTIONS section of a stochastics file
prsdpc	Read the PIECEWISE form of the DISTRIBUTIONS section of a stochastics file
prsdsc	Read the SCENARIOS form of the DISTRIBUTIONS section of a stochastics file

Table 8. Routines that read specific sections in the standard format

Subroutine	Function
prfmat	Read a matrix from a specified file
prfnam	Read a list of names from a specified file
prfvec	Read a vector from a specified file
prssvc	
prfget	Read lines from a specified file until a noncomment line is encountered
pflush	Read lines from a specified file until a section header is encountered

Table 4. General input routines

Subroutine	Function
paroll	Reorder the contents of an array
pchsrc	Search a character hash table for a specified key
pctsrc	Search a character simple list for a specified key
pntsrc	
pntst1	Insert a specified key in the proper position in an ordered numeric simple list
pntst2	
pchash	Hash a string of eight characters
pckbnd	Check a name against a set of bounds
pireal	Convert a character string to the real number it represents
pirint	Convert a character string to the integer it represents
pstrip	Remove leading blanks from a character string
pupper	Convert lower case letters in a character string to their upper case equivalents

Table 10. Miscellaneous utilities

Caveats

Although we have attempted to provide a comprehensive set of I/O handlers as well as several useful data structure manipulation routines, we realize that our collection is incomplete. We hope to enlarge it in the next few months. Furthermore, while a programmer will find it possible to read data expressed in the standard input format using some combination of our utilities, he or she may also discover mistakes in the logic of some routines. Our library is woefully untested. In most cases, however, it should prove a simple task to correct any errors. Finally, we would like to remind the reader that this paper is a proposal and that nothing is graven in stone.

Utility Descriptions

We now give a short summary of each of the utilities. We describe the basic function of each routine, specify the options that are available, and discuss how each routine might be used. A detailed description of the parameters taken by each routine and their meanings appears in comments to the source code, which is available from IIASA. Interested parties should contact

Project Secretary
SDS/ADO
International Institute for Applied Systems Analysis
A-2361 Laxenburg
Austria

ADCMAT

This subroutine generates the packed representation of a matrix of full rank, i.e., removes elements that are zero. The caller may direct ADCMAT to treat as zero any element whose absolute value is less than a specified tolerance.

ADCVEC

This subroutine generates the packed representation of a vector of full rank, i.e., removes elements that are zero. The caller may direct ADCVEC to treat as zero any element whose absolute value is less than a specified tolerance.

ADMKAX

This subroutine produces arrays to permit access to row vectors in a packed matrix, i.e., generates the augmented pointers, augmented indices, and map arrays.

ADXMAT

This subroutine generates a matrix of full rank from a packed matrix, i.e., restores elements that are zero.

ADXVEC

This subroutine generates a vector of full rank from a packed vector, i.e., restores rows or columns containing zero.

PAROLL

This subroutine adjusts the contents of one, two, or three arrays so that a given element moves to the top of the array(s) while the relative order of the array entries remains unchanged. It is used by other library routines to ensure that entries in stochastic vectors appear in strict row order.

PCHASH

This subroutine calculates the initial index and offset used by the method of double hashing for an eight character key.

PCHSRC

This subroutine is adapted from L. Nazareth's HASH. PCHSRC searches a hash table for a given eight character key. It uses double hashing for collision resolution.

PCKBND

This subroutine searches a subset of a given list or table of names for a given eight character key. It is used by other library routines to search a list when a user wishes to restrict his or her attention to a certain submatrix or subvector in an input file.

PCTSRC

This subroutine searches a simple list for a specified eight character key.

PFLUSH

This subroutine reads lines from an input file until a noncomment line whose first character is nonblank is found. It returns the contents of that line to the caller.

The caller may direct this subroutine to convert any lower case characters in columns 1 through 12, 15 through 22, and 40 through 47 (the code field, the three name fields, and the first and fourth columns) to their upper case equivalents.

This subroutine essentially skips to the next section header in the input when called. It can therefore be used to bypass unused sections in the stochastics or core files or to "flush" any lines that may remain in a section after the appropriate data has been read.

PIREAL

This subroutine converts the first sequence of nonblank characters it finds in a given string into the real number that sequence represents. The sequence must contain a decimal point and only the first 15 characters in the sequence are significant. PIREAL removes the character sequence from the beginning of the string.

PIRINT

This subroutine converts the first sequence of nonblank characters it finds in a given string into the integer value that sequence represents. The sequence must not contain a decimal point and only the first 10 characters in the sequence are significant. PIRINT removes the character sequence from the beginning of the string.

PNTSRC

This subroutine searches for a given key in an ordered numeric simple list using a binary search. The list is assumed to be in ascending order. It is used by other routines in the library to determine whether a certain name has or has not been enountered previously in the input file.

PNTST1 and PNTST2

These subroutines insert a given key into a simple list using insertion sort. The routines assume the list is in one-to-one correspondence with another array and adjust this array so that the values remain strictly associated with the list entries (the associated array is assumed to be double precision in PNTST1 and half integer in PNTST2).

PNTST1 and PNTST2 are used by other routines in the library to ensure that the contents of packed vectors appear in strict row order.

PRCBOU

This subroutine is adapted from M. Saunders' MINOS and L. Nazareth's PREADB. PRCBOU reads the BOUNDS section of a core file. It assumes that the BOUNDS section header has been read by the calling routine. It terminates when it encounters a noncomment card or line with a character in the first column or if it encounters end of file. Values in the numeric fields *must* contain a decimal point in order to be interpreted properly.

The caller provides PRCBOU a list of names. Each time PRCBOU reads a column name, it searches this list. If the column name is not in the list, an error is generated.

Several options are available:

- The caller may specify that the acceptable column names are in a hash table rather than in a simple list.
- The caller may direct PRCBOU to take its first input from a character string rather than from the source file.

- The caller may direct PRCBOU to enter only the bounds on some set of columns into the arrays that hold them.
- The caller may inhibit output of diagnostic messages, of the bounds themselves, or of both.

Several checks are performed on the input data.

PRCROW

This subroutine is adapted from M. Saunders' MINOS and L. Nazareth's PREADR. PRCROW reads the ROWS section of a core file. It assumes that the ROWS section header has been read by the calling routine. It terminates when it encounters a noncomment card or line with a character in the first column or if it encounters end of file. PRCROW ignores 'D' type rows.

There are several options available:

- The caller may direct PRCROW to construct a simple list or to construct a hash table to hold the row names.
- The caller may direct PRCROW to take its first input from a character string rather than from the source file.
- The caller may inhibit output of diagnostic messages, of the row names and types themselves, or of both.

Several checks are performed on the input data.

PRFGET

This subroutine reads lines from an input file until a noncomment, nonblank line is found. It returns the contents of the line to the caller.

The caller may direct this subroutine to convert any lower case characters in columns 1 through 12, 15 through 22, and 40 through 47 (the code field, the three name fields, and the first and fourth columns) to their upper case equivalents.

This subroutine essentially returns the next "interesting" line in the input file.

PRFMAT

This subroutine is adapted from M. Saunders' MINOS and L. Nazareth's PREADC. PRFMAT reads matrix entries into a packed matrix. It can be used to read the COLUMNS section of the core file, the DETERMINISTIC and STOCHASTIC forms of the RECOURSE section of the stochastic file, and, in conjunction with ADMKAX, the STOCHASTIC form of the TECHNOLOGY section of the stochastics file. It assumes that the header card of the current section has been read by the calling routine. It terminates when it encounters a noncomment card or line with a character in the first column or if it encounters end of file.

Only nonzero elements of the array are entered into the packed matrix and values in the numeric fields *must* contain a decimal point in order to be interpreted properly.

The caller passes PRFMAT a list of names. Each time PRFMAT reads a row name, it searches this list. If the row name is not in the list, an error is generated.

Several options are available:

- The caller may direct PRFMAT to construct a simple list or to construct a hash table to hold the column names.
- The caller may specify that the acceptable row names are in a hash table rather than in a simple list.
- The caller may direct PRFMAT to take its first input from a character string rather than from the source file.
- The caller may direct PRFMAT to read only a fixed number of columns from the source file.
- The caller may direct PRFMAT to enter only the elements for some set of rows into the packed matrix.
- The caller may direct PRFMAT to treat all values whose absolute value is less than or equal to some tolerance as zero.
- The caller may direct PRFMAT to enter all values it encounters, including zeros, into the packed matrix.
- The caller may inhibit output of diagnostic messages, of the matrix contents themselves, or of both.

Several checks are performed on the input data.

PRFNAM

This subroutine reads a list of names from a file into a simple list. It assumes that there is one name per line, in the first name field (columns 5 through 12). It terminates when it encounters a noncomment card or line with a character in the first column or if it encounters end of file. It can be used to read the CORE form of the TECHNOLOGY and RECOURSE sections of the stochastics file.

Several options are available:

- The caller may direct PRFNAM to construct a hash table rather than a simple list.
- The caller may specify that a name must be unique or that a name may appear more than once in the source file.
- The caller may pass PRFNAM a set of names, specifying that every name in the source file must appear in this set.
- The caller may direct PRFNAM to take its first input from a character string rather than from the source file.
- The caller may direct PRFNAM to read only a fixed number of names from the source file.
- The caller may inhibit output of diagnostic messages, of the names themselves, or of both.

Several checks are performed on the input data.

PRFVEC

This subroutine is adapted from M. Saunders' MINOS and L. Nazareth's PREADC. PRFVEC a row or column vector from a file into a packed vector. It assumes that the row or column names appear in the second and/or third name fields (columns 15 through 22 and 40 through 47, respectively) and that the associated values are in the first and second numeric fields (columns 25 through 36 and 50 through 61, respectively). The subroutine terminates when it encounters a noncomment card or line with a nonblank character in the first column or if it encounters end of file. It can be used to read the RHS and

RANGES sections of the core file and serves as the basis for PRFMAT.

Only nonzero elements of the vector are entered into the packed vector. and values in the numeric fields *must* contain a decimal point in order to be interpreted properly.

The caller passes PRFVEC a list of names. Each time PRFVEC reads a row or column name, it searches this list. If the row or column name is not in the list, an error is generated.

Several options are available:

- The caller may specify that the list of acceptable row or column names is in a hash table rather than in a simple list.
- The caller may direct PRFVEC to take its first input from a character string rather than from the source file.
- The caller may direct PRFVEC to read lines until the contents of the first name field (columns 5 through 12) change.
- The caller may pass PRFVEC a number of pointers that define a subset of the list of acceptable row or column names and direct PRFVEC to enter only elements from those rows or columns into the packed vector.
- The caller may direct PRFVEC to treat all values whose absolute value is less than or equal to some tolerance as zero.
- The caller may direct PRFVEC to enter all values it encounters, including zeros, into the packed vector.
- The caller may inhibit output of diagnostic messages, of the vector contents themselves, or of both.

Several checks are performed on the input data.

PRSDIS

This subroutine reads the DISCRETE form of a standard format DISTRIBUTIONS section into a stochastic vector. It terminates when it encounters a non-comment card or line with a character in the first column or if it encounters end of file. Values in the numeric fields *must* contain a decimal point in order to be interpreted properly and the values specified for a given row must be distinct.

The caller passes PRSDIS a list of names. Each time PRSDIS reads the name of a row in the stochastic vector, it searches this list. If the row name is not in the list, an error is generated.

Several options are available:

- The caller may specify that the acceptable row names are in a hash table rather than in a simple list.
- The caller may direct PRSDIS to take its first input from a character string rather than from the source file.
- The caller may direct PRSDIS to enter only the elements for some set of rows into the stochastic vector.
- The caller may provide a tolerance that specifies how close the sum of the probabilities for the values that a given row may assume must be to unity.
- The caller may inhibit output of diagnostic messages, of the vector contents themselves, or of both.

Several checks are performed on the input data.

PRSDPC

This subroutine reads the PIECEWISE form of a standard format DISTRIBUTIONS section into a stochastic vector. It terminates when it encounters a noncomment card or line with a character in the first column or if it encounters end of file. Values in the numeric fields MUST contain a decimal point in order to be interpreted properly.

The caller passes PRSDPC a list of names. Each time PRSDPC reads the name of a row in the stochastic vector, it searches this list. If the row name is not in the list, an error is generated.

Several options are available:

- The caller may specify that the acceptable row names are in a hash table rather than in a simple list.
- The caller may direct PRSDPC to take its first input from a character string rather than from the source file.
- The caller may direct PRSDPC to enter only the elements for some set of rows into the stochastic vector.
- The caller may provide a tolerance that specifies how close the sum of the probabilities for the ranges for a give row must be to unity.
- The caller may inhibit output of diagnostic messages, of the vector contents themselves, or of both.

Several checks are performed on the input data.

PRSDSC

This subroutine reads the SCENARIOS form of a standard format DISTRIBUTIONS section into a stochastic vector. It terminates when it encounters a noncomment card or line with a character in the first column or if it encounters end of file. The values in the numeric fields *must* contain a decimal point in order to be interpreted properly.

The caller passes PRSDSC a list of names. Each time PRSDSC reads the name of a row in the stochastic vector, it searches this list. If the row name is not in the list, an error is generated.

Several options are available:

- The caller may specify that the acceptable row names are in a hash table rather than in a simple list.
- The caller may direct PRSDSC to take its first input from a character string rather than from the source file.
- The caller may direct PRSDSC to enter only the elements for some set of rows into the stochastic vector.
- The caller may provide a tolerance that specifies how close the sum of the probabilities for the scenarios must be to unity.
- The caller may inhibit output of diagnostic messages, of the vector contents themselves, or of both.

Several checks are performed on the input data.

PRSSVC

This subroutine is a generic input utility for the stochastics file. It can be used to read the OBJECTIVES section and serves as a building block for routines that read the DISCRETE and SCENARIOS forms of the DISTRIBUTIONS section. The subroutine terminates if it encounters a noncomment card or line with a

nonblank character in the first column or if it encounters end of file. Values in the numeric fields *must* contain a decimal point in order to be interpreted properly.

PRSSVC supports a number of options that allow it to read several sections of the standard input format. Among them:

- The caller may direct PRSSVC to enter the data it reads into a packed vector or into the arrays for a stochastic vector. The former is required to read the OBJECTIVES section of the stochastics file, the latter to to read the DISTRIBUTIONS section of the stochastics file.
- The caller may direct PRSSVC to ignore lines whose first numeric value has an absolute value less than or equal to a specified tolerance.
- The caller may direct PRSSVC to read lines until the contents of any combination of the code field (columns 2 and 3), the first name field (columns 5 through 12), and the second name field (columns 15 through 22) change from what they were when the first valid line was read. Routines to read the sections listed below can profit from using this capability with the combinations shown:

Section (Form)	Fields
DISTRIBUTIONS (SCENARIOS)	code, first name
DISTRIBUTIONS (DISCRETE)	first name, second name
OBJECTIVES (LINEAR)	first name
OBJECTIVES (PIECEWISE)	first name

- The caller may pass PRSSVC a list of names. In this case, each time PRSSVC reads a line, it searches this list to see whether the contents of the second name field on the line (columns 15 through 22) appear in the list. If not, an error is generated. The caller may specify that the list of acceptable names is in a hash table rather than in a simple list. This option simplifies the task of reading the OBJECTIVES section and the SCENARIO form of the DISTRIBUTIONS section.
- The caller may pass PRSSVC a number of pointers that define a subset of the list of acceptable names and direct PRSSVC to enter only values associated with those names into the appropriate structure.
- The caller may direct PRSSVC to enter both values that appear on a line into the values arrays or to enter only the first value into the first values array. The former is useful when reading the DISCRETE form of the DISTRIBUTIONS section and the PIECEWISE form of the OBJECTIVES section and the latter is useful when reading the SCENARIOS form of the DISTRIBUTIONS section and the LINEAR form of the OBJECTIVES section.
- The caller may specify that the values appearing in the first numeric field must be unique. This option is used to verify that the values for the rows in the DISCRETE form of the DISTRIBUTIONS section are unique. In this case, the values that appear in the first numeric field are sorted in ascending order when returned. For this reason, this option cannot be specified with a list of acceptable names or with packed vector data structures (this option is ignored if conflicting options are specified).
- The caller may direct PRSSVC to take its first input from a character string rather than from the source file.
- The caller may inhibit output of diagnostic messages, of the input data itself, or of both.

Several checks are performed on the input data.

PSTRIP

This subroutine removes leading blanks from a given string and returns the modified string to the caller.

PUPPER

This subroutine converts any lower case characters in the code field, the three name fields, and the first and fourth columns (columns 1 through 12, 15 through 22, and 40 through 47) of a given line to their upper case equivalents.

Visions of Utopia

The routines presented here are, alas, rough drafts only. Readers are encouraged to make whatever modifications (or corrections) they may need. Work will continue, however, in an effort to expand the range of the modules, improve their ease of use, eradiacate bugs, and standardize calling conventions.

References

ANSI (American National Standards Institute), (1978) **American National Standard Programming Language FORTRAN (ANSI X3.9-1978 FORTRAN 77)**

IBM (International Business Machines, Inc.), (1972) **Mathematical Programming Subsystem - Extended (MPSX) and Generalized Upper Bounding (GUB) Program Description** document number SH20-0968-1 (document number 830056 in the IIASA library)

Knuth, D.E. (1973), **The Art of Computer Programming, Volume 3/Sorting and Searching,** Addison Wesley

Murtagh, B.A. and M.A. Saunders (1977), **MINOS - A Large-Scale Nonlinear Programming System (For Problems with Linear Constraints) - User's Guide,** Technical Report SOL 77-9, Systems Optimization Laboratory, Department of Operations Research, Stanford University

Nazareth, L. (1982), Implementation Aids for Optimization Algorithms that Solve Sequences of Linear Programs by the Revised Simplex Method, IIASA Working Paper WP-82-107